Apple IIGS
#83: Resource Manager Stuff

Revised by: Matt "Even less of a middle name" Deatherage May 1992
Written by: Dave Lyons May 1990

This Technical Note answers your miscellaneous Resource Manager questions.

Changes since December 1991: Added several notes pertaining to System Software 6.0 and a note about making Resource Manager calls from a resource converter. Added new discussion about how "changed" is really a resource attribute.

UniqueResourceID

In System Software 5.0.4 and earlier, calling `UniqueResourceID` with an `IDRange` value of $FFFF does not work reliably. It sometimes returns a system-range ID ($07FFxxxx) if there are already system-range resources of the specified type present in the current search path.

If you are using a development utility that generates resource IDs using `UniqueResourceID`, check the results to make sure no system-range resource IDs are being used by accident. This problem is fixed in System Software 6.0.

What SetCurResourceFile Does

`SetCurResourceFile` is documented in Chapter 45 of the *Apple IIGS Toolbox Reference*, Volume 3 (see especially "Resource File Search Sequence" near the beginning of the chapter).

This explanation might make you think `SetCurResourceFile` rearranges the search path, but it does not; instead, it just makes searches start at a different place in the path. `SetCurResourceFile` is useful for controlling what resource files are searched, not for changing the search order.

How the Toolbox Uses Resources as Templates

The toolbox uses several types of resources as templates for creating other objects. Examples include `rControlList`, `rControlTemplate`, and `rWindParam1`. The toolbox automatically releases these resources from memory as soon as it is through with them, so there is no need to create your template resources with special purge levels in an effort to free more memory. It is not a problem.

Using Resources From Window Update Routines

In System Software 6.0 and earlier there is no special code to set the current resource application when the system calls an application window update routine (See Apple IIGS Technical Note #71 for notes on NDAs and the current resource application).

To avoid a situation where a window update routine cannot get needed resources, obey the following rules:

1. Application window update routines must **either** (a) assume that the resource application has the same value it had when the window was created, or (b) save, set, and restore the current resource application, using `GetCurResourceApp` and `SetCurResourceApp`.

2. NDAs that start the Resource Manager must not call application window update routines, and they must not cause application window update routines to be called (for example, if an NDA calls `TaskMaster` to handle a modal dialog or movable modal dialog, the `tmUpdate` bit in `wmTaskMask` must be off).

CurResourceApp in InfoDefProcs and Custom Windows

The current resource application has no guaranteed value when an information bar definition procedure or custom window definition procedure gets control. These must always save, set, and restore the current resource application using `GetCurResourceApp` and `SetCurResourceApp`.

StartUpTools Opens Resource Forks Read-Only

When `StartUpTools` opens your application's resource fork, by default it opens it with read-only access. If your application needs to make changes to the resources on disk in System Software 5.0.4 and earlier, you need to close the fork and reopen it with read and write access. To close it, use `GetCurResourceFile` and `CloseResourceFile`; to reopen it, use `LGetPathname2` and `OpenResourceFile`.

Note: You must update the `resFileID` field in the `StartStop` record if you close and reopen your resource fork. `CloseResourceFile` disposes the handles of any resources in memory from the file you're closing, so you must call `DetachResource` on any resources you need to keep. (If you pass an `rToolSTartup` resource to `StartUpTools`, the system detaches it for you automatically.)

In System Software 6.0 and later, setting bit 3 ($0008) of the `startStopRefDesc` tells the Tool Locator to open your resource fork with all allowed permissions instead of with just read permission.

Calling StartUpTools From a Shell Application (File Type $B5, EXE)

In System Software 5.0.4 and earlier, `StartUpTools` tries to open the current application's resource fork. It determines the pathname of the "current application" by examining prefix `9:` and making a `GET_NAME` GS/OS call, but do not assume it will always construct the pathname this way. If you call `StartUpTools` from a shell application and expect it to open your EXE file's resource fork, you will be disappointed.

If GS/OS has launched your application, life is good—usually, though, a shell has loaded your shell application directly, so `GET_NAME` returns the name of the shell instead of the name of your application file.

To open your shell file's resource fork, call `ResourceStartUp`, get the pathname by calling `LGetPathname2` on your user ID, and pass the pathname to `OpenResourceFile`. `StartUpTools` uses this strategy all the time in System Software 6.0 and later, meaning you don't have to.

What's NIL in a Resource Map?

The resource maps for open resource files are kept in memory, and the structure is defined in chapter 45 of *Apple IIGS Toolbox Reference*, Volume 3.

The `resHandle` field of a resource reference record (`ResRefRec`) is defined as "Handle of resource in memory. A `NIL` value indicates that the resource has not been loaded into memory." In this case, `NIL` means that the middle two bytes of the four-byte field are zero. In other words, a `NIL` entry in the resource map may have a non-zero value in the low-order byte.

LoadResource and SetResLoad(FALSE)

When you call `LoadResource` on a locked or fixed resource and `SetResLoad` is set to `FALSE`, you may get Memory Manager error $0204 (`lockErr`), because the Resource Manager tries to allocate a locked or fixed zero-length handle, which the Memory Manager does not permit.

Adjusting the Search Depth

If you wish to add some resource files to the beginning of a resource search path and adjust the depth so that the end point of the search is unchanged, it's tempting to use `SetResourceFileDepth(0)` to get the current depth, add one, and set this new depth with `SetResourceFileDepth`.

The problem is that the search depth is often -1 ($FFFF), meaning "search until the end of the chain." If you add your adjustment to -1, you do not usually get the intended effect. A solution is just to check for $FFFF and not adjust the depth in that case.

CurResourceApp after ResourceShutDown

After a `ResourceShutDown` call, the current resource application is always $401E. (The Resource Manager starts itself up at boot time with its own memory ID, $401E. Do not ever call ResourceShutDown while the current resource application is $401E.)

Restoring the CurResourceApp

If you need to start up and shut down the Resource Manager without disturbing the current resource application, call `GetCurResourceApp` **before** `ResourceStartUp`, and call `SetCurResourceApp` to restore the old value **after** `ResourceShutDown`.

It does not help to call `GetCurResourceApp` after `ResourceStartUp`, since the application just started up is always the current resource application.

Shell programs which start the Resource Manager need to call `SetCurResourceApp` after regaining control from a subprogram (for example, an EXE file) which may have started and shut down the Resource Manager, leaving the current resource application set to $401E instead of the shell's ID.

Shell programs that do not start the Resource Manager have nothing to worry about. In this case the current resource application is normally $401E, so when a subprogram calls `ResourceShutDown` life is still wonderful.

What Information is Kept For Each Resource Application?

When you switch resource applications with `SetCurResourceApp`, that takes care of all the application-specific information the Resource Manager has.

There is no need to separately preserve the current resource file, the search depth, the `SetResourceLoad` setting, or any application resource converters that are logged in. All of this information is already recorded separately for each resource application.

"Changed" is a Resource Attribute

This seems obvious when first reading the documentation, but it has a consequence that isn't so obvious.

If you mark a resource as changed with `MarkResourceChanged` and later use `SetResourceAttr` to change that resource's attributes, you must include `resChanged` in the attributes you specify or the Resource Manager does **not** still know the resource has changed.

This means you can undo a `MarkResourceChanged` call, but it also means you need to preserve the `resChanged` bit across `SetResourceAttr` calls if you don't want to accidentally achieve the same effect.

The Resource Manager clears the `resChanged` attribute when a resource is written to disk; the attribute indicates the data in memory is more recent than what's on disk. Normally, adding a resource with `AddResource` sets this bit because the resource isn't actually written to disk until the resource file is updated.

However, if `AddResource` has to make the file longer (by extending the EOF), it writes the resource to disk immediately. This means that in some cases, a resource added with `AddResource` will be properly added but the `resChanged` attribute will **not** be set. Don't be confused if this happens to you.

Making Resource Manager Calls From Resource Converters

Don't. This would be a first-class example of reentrancy, and the Resource Manager is not reentrant in any class.

Who Owns Handles Passed to AddResource?

When you pass a handle to `AddResource`, the Resource Manager is responsible for the handle unless `AddResource` returns an error. Once you call `AddResource`, the handle belongs to the Resource Manager and you must treat it like you would the handle to any other resource.

Named Resource Bugs in System Software 6.0

The new-for-6.0 Resource Manager function `RMFindNamedResource` compares the resource name you requested to named resources incorrectly. The comparison algorithm doesn't compare the lengths of the strings before starting to compare the characters. This means, for example, that if you request a resource named "Raymond" and the Resource Manager encounters a named resource named "Raymond" first, it will return the resource named "Raymond" instead. This anomaly also affects the HyperCard IIGS named-resource XCMD callback functions, even though they don't use the Resource Manager's named-resource calls.

This anomaly also affects `RMLoadNamedResource`, which calls `RMFindNamedResource`.

Debugging Information

The following information is provided for your convenience during program development. It allows you to check exactly what user IDs are using the Resource Manager, what files are in their search paths, and what resource converters are logged in.

Do not depend on this information in your program; it is subject to change in future versions of the Resource Manager.

All the Resource Manager's data structures are rooted in the Resource Manager tool set's Work Area Pointer (WAP). To get the Resource Manager's WAP, call `GetWAP` (in the Tool Locator) with `userOrSystem` = $0000 and `tsNum` = $001E.

The WAP value is a handle to the Resource Manager's block of global data. Several interesting areas in this block are listed below.

+$0A2	curApp	**Word**	Offset into the globals block of the current resource application's Application Record.
+$2B0	sysFile	**Long**	Handle of system file map, or NIL if none.
+$2B4	sysConvertList	**Long**	Handle of system converter list, or NIL if none.
+$2B8	appList	**20*n bytes**	List of Application Records (20 bytes each).

Each Application Record has this format:

+000	appFlag	**Word**	0=entry available, 1=entry used, $FFFF = end of array.
+002	appID	**Word**	User ID of application.
+004	appFiles	**Long**	Handle of application's first resource map, NIL=none.
+008	appCur	**Long**	Handle of application's current resource map, NIL=none.
+012	appConverters	**Long**	Handle of application's converter list, NIL=none.
+016	appReadFlag	**Word**	1=read resources, 0=don't read (`SetResourceLoad`).

+018	appFileDepth	**Word**	Number of files to search in this path.

Converter lists have this format:

+000	n	**Word**	Number of entries in the table (entries can be unused).
+002	theConverters	**6*n bytes**	List of converter entries (6 bytes each).

Each Converter entry has this format:

+000	resType	**Word**	Resource type for this converter ($0000 for unused entry).
+002	convAddress	**Long**	Address of resource converter.

The format for a resource map is described starting on page 45-17 of *Apple IIGS Toolbox Reference*, Volume 3.

Remember, don't depend on this information in your application; use it during debugging, and use it to write debugging utilities.

Further Reference

- *Apple IIGS Toolbox Reference,* Volume 3
- Apple IIGS Technical Note #71, DA Tips and Techniques